AT&T IDS NO: 2000-0542

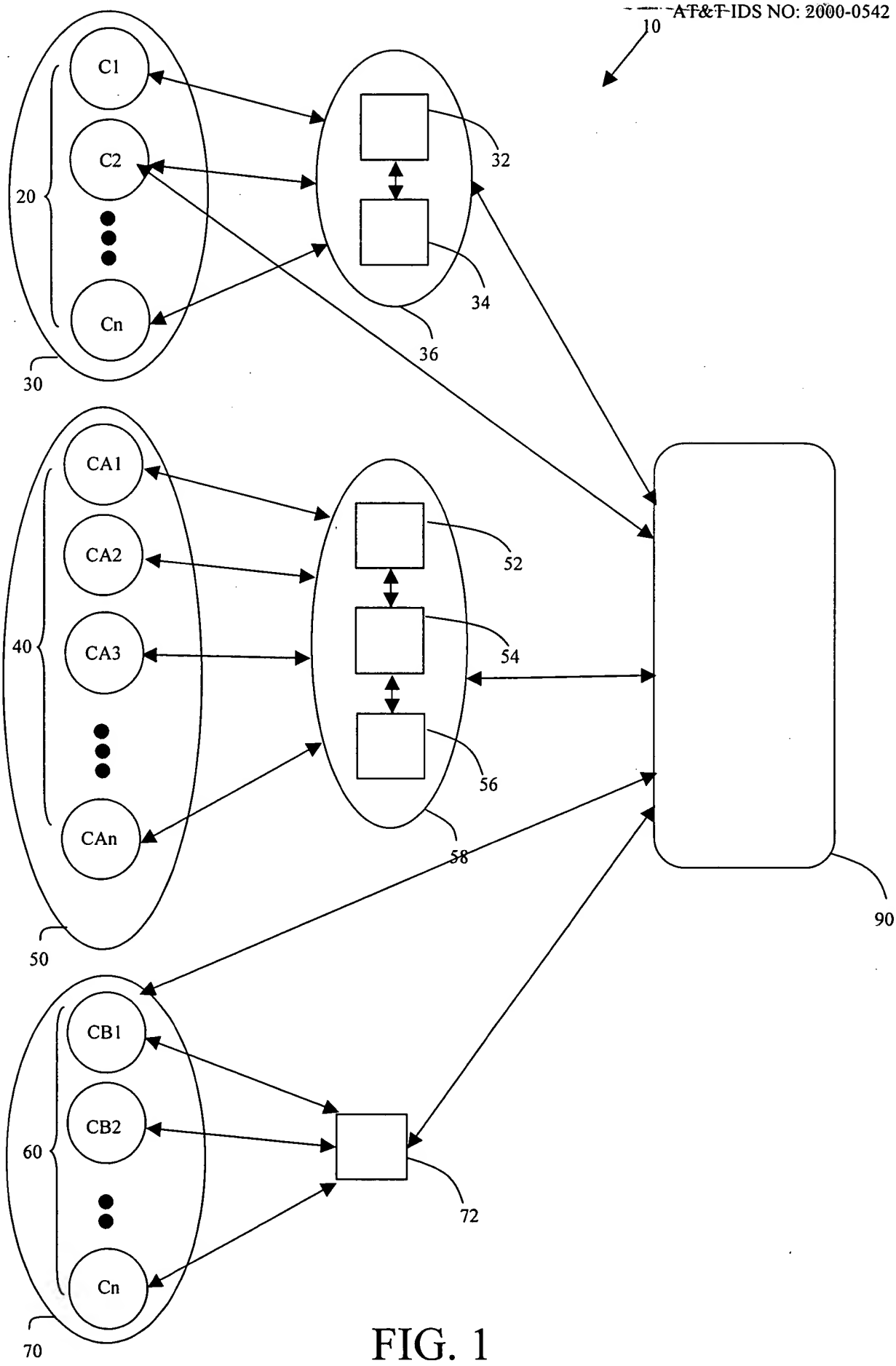

FIG. 1

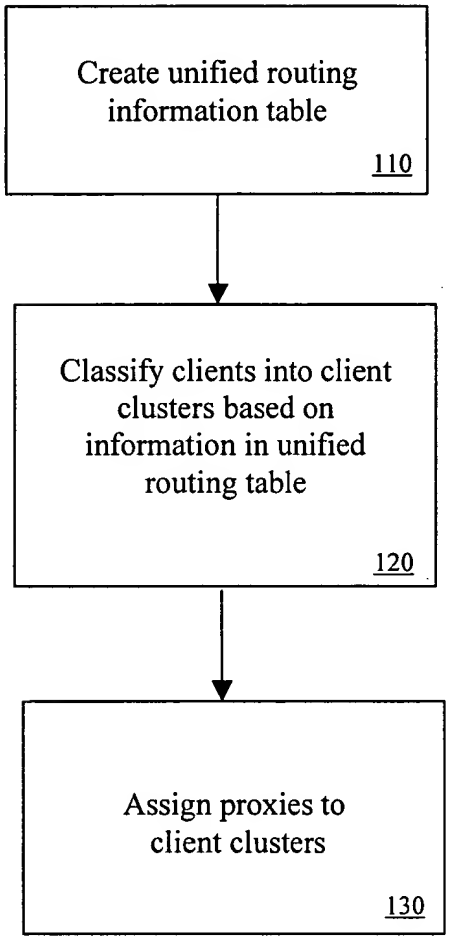
Create unified routing information table
110
Classify clients into client clusters based on information in unified routing table
120
Assign proxies to client clusters
130

FIG. 2

160

Routes		AS path
134.87.2.0/24	BCnet	6509 271
134.117.0.0	Carleton University	6509 10786
137.82.0.0	University of British Columbia	6509 271
137.122.0.0	University of Ottawa	6509 10786

170 180 190

FIG. 3

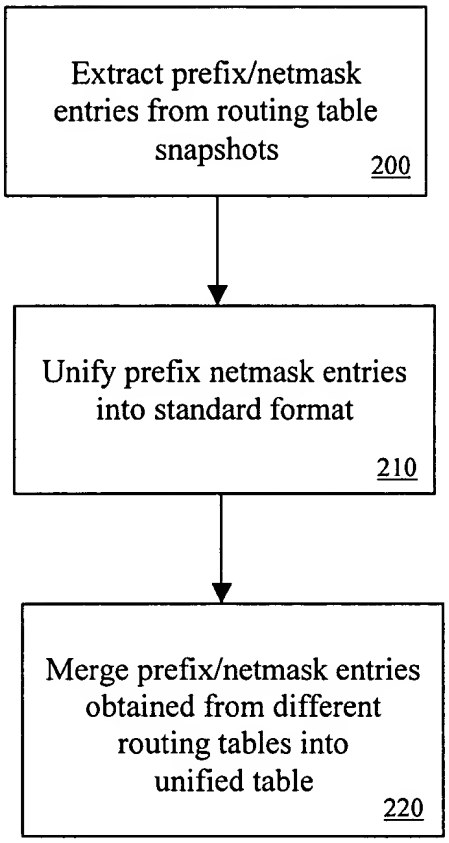
Extract prefix/netmask entries from routing table snapshots
200
Unify prefix netmask entries into standard format
210
Merge prefix/netmask entries obtained from different routing tables into unified table
220

FIG. 4

	230	240	250
Formats	x1.x2.x3.x4/k.1k.2k.3k.4	x1.x2.x3.x4/l	x1.x2.x3.0
Routing Tables	MAE-EAST MAE-WEST PACBELL PAIX	ARIN AT&T CANET NLANR VBNS	CANET
Examples	193.1/255.255 193.0.128/255.255.192	128.148.0.0/16	130.15.0.0
Unification	193.1/255.255 193.0.128/255.255.192	128.148.0.0/255.255	130.15.0.0/255.255 192.75.72/255.255.255

FIG. 5

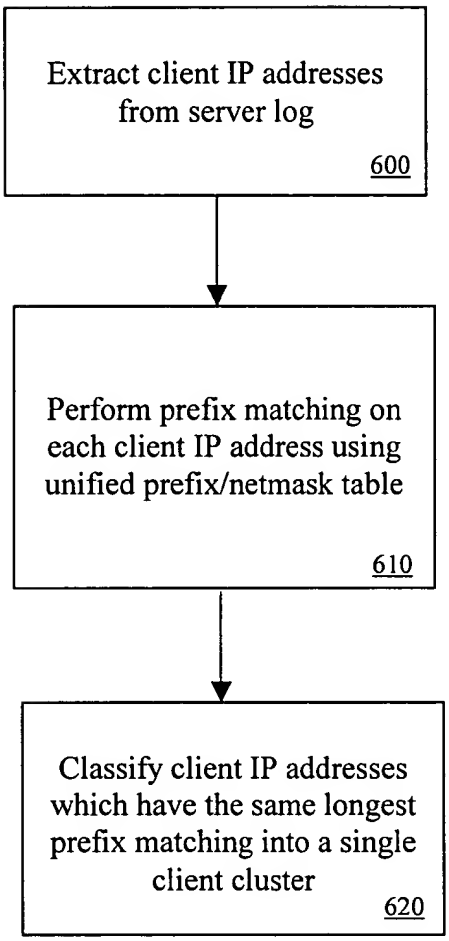
Extract client IP addresses from server log
600
Perform prefix matching on each client IP address using unified prefix/netmask table
610
Classify client IP addresses which have the same longest prefix matching into a single client cluster
620

FIG. 6

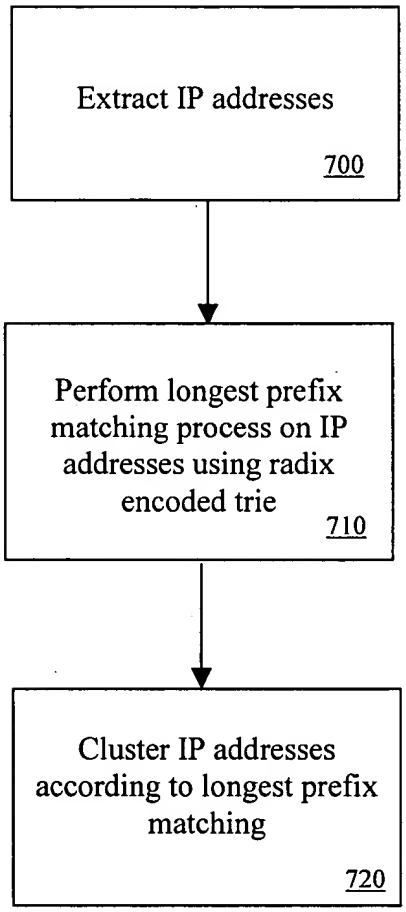
Extract IP addresses
700
Perform longest prefix matching process on IP addresses using radix encoded trie
710
Cluster IP addresses according to longest prefix matching
720

FIG. 7

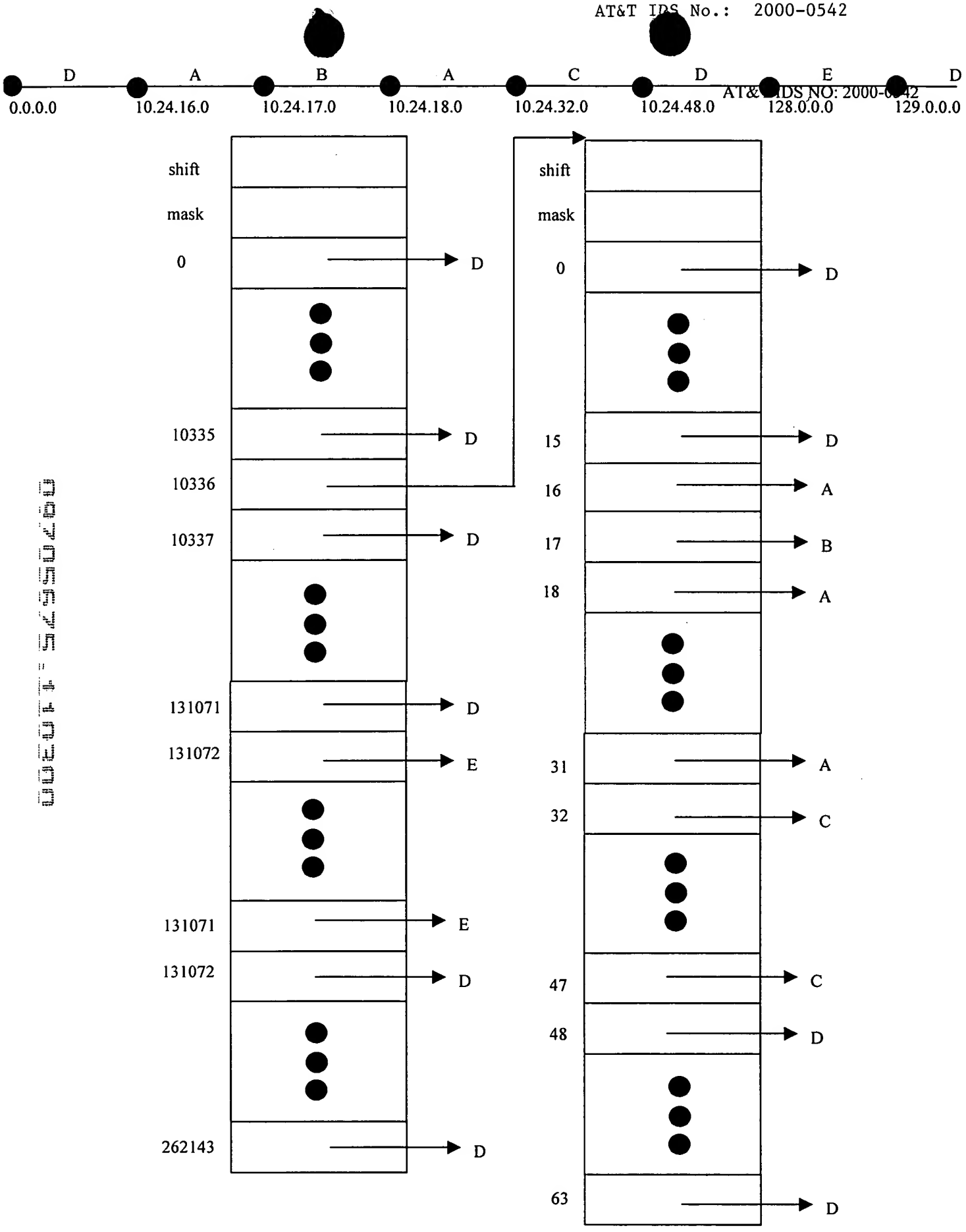
D
A
B
A
C
D
E
D
0.0.0.0
10.24.16.0
10.24.17.0
10.24.18.0
10.24.32.0
10.24.48.0
128.0.0.0
129.0.0.0
shift
mask
0
D
10335
D
10336
10337
D
131071
D
131072
E
131071
E
131072
D
262143
D
shift
mask
0
D
15
D
16
A
17
B
18
A
31
A
32
C
47
C
48
D
63
D

FIG. 8

matching code for the <S,R,H> values generated by the first part of the algorithm

```
/*
 * multi-level retrie longest prefix match
 *
 * input arguments:
 *
 *    S     top level shift
 *    R     retrie internal nodes
 *    H     retrie leaf nodes (next hop index)
 *    addr  IP address to match
 *
 * return:
 *
 *    0     no match
 *    >0    next hop index
 */

int
lpmatch(uint8 S, uint32* R, uint8* H, uint32 addr)
{
      uint32      b;
      uint32      x;

      x = R[addr >> S];
      while (b = x >> 27)
      {
            b = (x & ((1<<26)-1)) + ((addr>>(S-=b)) & (1<<b)-1));
            if (x & (1<<26))
            {
                  x = H[b];
                  break;
            }
            x = R[b];
      }
      return x;
}
```

FIG. 9a

```
/*
 * 2-level retrie longest prefix match
 *
 * input arguments:
 *
 *    S     top level shift
 *    R     retrie internal nodes
 *    H     retrie leaf nodes (next hop index)
 *    addr  IP address to match
 *
 * return:
 *
 *    0     no match
 *    >0    next hop index
 */

int
lpmatch2(uint8 S, uint32* R, uint8* H, uint32 addr)
{
      uint32      x;

      x = R[addr >> S];
      return H[(x & ((1<<26)-1)) + ((addr & ((1<<S)-1)) >> (S - (x>>27)))];
}
```

FIG. 9b